VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2019 Matouš Mudřík

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Matouš Mudřík**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: ITA spol. s r.o. Ostrava
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Michal Radecký, Ph.D.**

Konzultant bakalářské práce: Ing. Daniel Hajduk, Ph.D.

Datum zadání: 01.09.2018

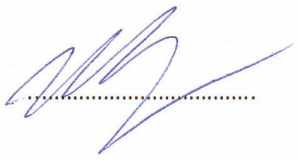
Datum odevzdání: 30.04.2019

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 25. Dubna 2019



Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

V Ostravě 25. dubna 2019

.....

ITA společnost s r.o.
Martinská 6, 709 00 Ostrava
IČO: 15503933
DIČ: CZ15503933

Rád bych na tomto místě poděkoval Tomáši Mockovi, Františkovi Krupovi, Danieli Hajdukovi a Zuzaně Kavkové, kteří mi s prací pomohli a bez nichž by tato práce nevznikla.

Abstrakt

Obsahem této bakalářské práce je odborná praxe, která se odehrávala ve firmě ITA technology and software v Ostravě.

Moje práce spočívala v optimalizaci starých programů napsaných ve Fortranu. Ve většině případů se jednalo o kompletní přepsání starého programu se starým kódem do novějšího, který vyhovoval danému řešení lépe. Dále úpravách těchto programů a doplnění je o vlastnosti, které mně byly zadány. Také naprogramování Windows aplikací v MFC knihovně, která sloužila jako prezentační program pro matematické výpočty související s válcováním. K těmto úkolům byly potřebné pokročilé znalosti Fortranu, C++ a základy řešení matematických rovnic.

Klíčová slova: MFC, Fortran, C++, Visual Studio, ITA technology and software

Abstract

The content of this bachelor thesis is professional practice, which is based on ITA technology and software in Ostrava.

My job was optimizing old programs written in Fortran. In most cases, this is the complete overwriting of the old program into the newer one that fits that solution. And then on editing these programs and adding features which was given to me. Also i had to program the window application in the MFC library, which served as a presentation program for mathematical calculations related to rolling. For these tasks, we have used advanced knowledge of Fortran, C++ and basics of solving mathematical equations.

Key Words: MFC, Fortran, C++, Visual Studio, ITA technology and software

Obsah

Seznam použitých zkratek a symbolů

MFC	–	knihovna v C++, která obsahuje podporu pro Windows aplikace
MPI	–	knihovna implementující stejnojmennou specifikaci (protokol) pro podporu paralelního řešení výpočetních problémů v počítačových clusterech
OOP	–	oběktově orientované programování
LPCSTR	–	dlouhý pointer na Constantu Tchar String
SDK	–	Software Development Kit - sada nástrojů pro vývoj softwaru
API	–	Application Programming Interface - soubor funkcí a postupů umožňujících vytvářet aplikaci
GUI	–	grafické uživatelské rozhraní (Graphical User Interface). Uživatelské rozhraní, které umožňuje ovládat počítač pomocí interaktivních grafických ovládacích prvků.

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Místo své bakalářské praxe jsem si vybral společnost ITA technology & software Ostrava, kde jsem odpracoval během studia, celkově přes 100 pracovních dnů.

1.1 Firma ITA

Společnost ITA technology & software je soukromá česká společnost založená v roce 1991 výzkumnými a vědeckými pracovníky Výzkumného ústavu strojírenského a metalurgického. Zabývá se moderními technologiemi válcování za tepla i za studena, dodává know-how a programová řešení významným dodavatelům válcovacích zařízení, technologií a řídících systémů. Mezi tyto firmy patří například Danieli, ConverTeam, Acos Vilares, Vitkovice Heavy Machinery.

Dále řeší technické problémy a technologické inovace na teplých a studených válcovacích tratích jako např. ArceloMittal Ostrava, Severstal Čerepovec, ArcelorMittal Vanderbijlpark, CSN Voltraredunda a Třinecké železárny Třinec.

Hlavními aktivitami je řešení technických problémů, zpracování studií, konzultační a poradenská činnost v oblasti technologií válcování za tepla i za studena. Dále vývoj nových, optimalizace a úpravy stávajících SW modulů řídících systémů válcovacích tratí, vývoj speciálních programů určených pro off-line simulace procesů válcování a ochlazování. A počítačové modelování procesů tváření a tepelného zpracování. (ITA spol. s.r.o. Udemy [14])

1.2 Praxe

Moje účast ve firmě se zakládala na optimalizaci a úpravách stávajících SW modulů řídících systémů válcovacích tratí a vývoj speciálních programů určených pro off-line simulace. První dny na praxi jsem se seznamoval s programem RollFlex a s jazykem Fortran, aniž bych měl nějaké předešlé zkušenosti s tímto jazykem. Pro orientaci v kódu, porozumění klíčovým slovům, logice programu a celé struktuře, bylo potřeba udělat jeho hluboký výzkum. Po proniknutí do této problematiky začala vlastní práce na projektu, která se skládala z několika částí.

1. Přepisování programů napsaných v jazyce Fortran do jazyka C++. Tato část obsahovala spoustu seznamovacích a vzdělávacích činností s jazykem.

2. Sloučení všech přeložených programů, které už byly otestované a funkční jako samostatné části.

3. Vytvoření prezentační programu. Jednalo se o demo verzi finálního projektu, který slouží k ukázce a předvedení programu. Obsahoval některé výpočty a především grafy, které zobrazovaly průběh působících sil na provalek a mírů protlačení materiálu.

4. Vytvoření nových funkcí programu pro jeho lepší využití.

5. Optimalizace programu. Jednalo se o odstranění všech zbytečných částí, zjednodušení kódu a zrychlení systému jako například odstranění duplicity kódu a upravení na třídy.

2 Použité jazyky

V této práci byly použity dva programovací jazyky a to C++ a Fortran. Pro porozumnění problematice je nutné si vysvětlit základy obou jazyků a porovnat je mezi sebou.

2.1 Seznamení s jazykem Fortran

Fortran je dnes používán pouze zřídka. Dle různých stupnic a měřítek použití jazyků se nachází přibližně na 30. pozici. I přesto je Fortran stále dominantní jazyk pro simulace fyzických systémů jako je astrofyzikální modelování hvězd a galaxií, hydrodynamické kódy, molekulární dynamika ve velkém měřítku. Je velmi účinný v oblasti vysoce výkonných výpočetních systémů, u nichž se v dnešní době používají nejčastěji C++ a "modern Fortran"(Fortran 90/95/03/08). Pro tyto dva jazyky byly vyvinuty Open MPI knihovny pro paralelní kód. Tudíž, chceme-li rychlý kód, který běží na mnoha procesorech, tak máme nejrozumnější možnosti C++ nebo Fortran. (Learning Fortran TutorialPoint [5])

2.2 Jazyk C++

C++ je rozšíření jazyka C. Je rozšířen o objektově orientované programování, které přináší nový přístup řešení programování velkých celků (projektů). Na rozdíl od procedurálního programování, které zdůrazňuje algoritmy, OOP klade důraz na data. Spíše než se snažit, aby problém vyhovoval procedurálnímu přístupu jazyka, OOP usiluje o to, aby jazyk vyhovoval problému. Myšlenka spočívá v návrhu datových forem, které odpovídají základním rysům problému. (Stephan Prata, Mistrovství v C++ [1]) (Learn advance C++ on Udemy [10])

2.2.1 OOP

Programování podle OOP představuje mnohem více než pouhé svázání dat a metod s definicí třídy. OOP například podporuje vytváření znovupoužitelného kódu, který nakonec může ušetřit mnoho práce. Skrývání informací pomocí modifikátorů zabezpečuje data proti nevhodnému přístupu. Polymorfismus umožňuje vytvářet vícenásobné definice operátorů a funkcí, přičemž souvislosti v kódu určují, která definice se má použít. Pomocí dědičnosti je možné odvozovat nové třídy od starých. Je zřejmé, že OOP zavádí mnoho nových pojmů a vyžaduje odlišný přístup k programování než procedurální programování. (Robert Lafore, OOP in Microsoft C++ [2])

2.2.2 Výraz class

Výraz class popisuje specifikaci formy dat a objekt představuje určitou datovou strukturu s konstruovanou podle dané třídy. Třída je uživatelský datový typ, který uchovává vlastní členy dat a členské funkce, ke kterým lze přistupovat a používat je za pomoci instance dané třídy. Jedná se o takový plán objektů. (From Beginner to Beyond on Udemy [12])

2.3 Fortran vs. C++

Pro moji práci bylo nejdůležitější zjistit rozdíly mezi Fortranem a C++. Na všech webových stránkách s rychlostními testy C++ vyhrává C++ nad Fortranem asi v 80% případů. (The Computer Language Benchmarks [8]) Oba jazyky se pohybovaly přibližně na stejných hodnotách. Z testů vyplývalo, že pokud se jedná o vícejádrový stroj, tím je C++ rychlejší. Další zjištěnou informací bylo, že procesy, které zahrnují časté čtení a zápis dat, jsou větší zátěží pro Fortran. Z výsledků vyplývalo, že C++ je ve většině případů rychlejší než Fortran. Pouze v několika málo případech byl o málo pomalejší, v některých případech naopak mnohonásobně rychlejší. Nemluvě o tom, že čím více jader, tím rychlejší (lepší) výsledky pro C++, což je další výhoda. Tabulka srovnání rychlosti.(Tab.1)

2.3.1 Proč Fortran

Vědci používají Fortran především proto, že byl vytvořen asi o 10-15 let dříve než jazyk C. Fortran byl publikován okolo roku 1957 naproti tomu C roku 1972. Dalším důvodem, proč se Fortran používá ještě dodnes je, že se vědci snaží minimalizovat množství kódování, a proto používají svůj starší kód, replikují ho, obnovují a předávají studentům. Může to být i tím, že C++ je mnohem komplexnější, a tím těžší jazyk na pochopení než Fortran. (More is diferent [9])

2.3.2 V čem spočívá jednoduchost Fortranu

Hlavním důvodem je, že Fortran není objektově orientovaný jazyk. Objektové programování je užitečné hlavně v rozsáhlejších softwarových projektech. Zabere více času na naprogramování. OOP obsahuje abstraktní pojmy jako jsou třídy a dědictví. Paradigma OOP se velmi liší od procesního paradigmatu používaného ve Fortranu. Fortran je založen na jednoduchém procesním paradigmatu, který je bližší tomu, co se vlastně děje v počítači "pod kapotou". Optimalizace objektově orientovaného kódu je z mého pohledu obtížnější než procesní kód.

Výhodou je, že proměnné ve Fortranu jsou standartně předávány odkazem a ne hodnotou. Kompilátor Fortranu automaticky optimalizuje průchod tak, aby byl co nejefektivnější. Pro fyziky, matematiky a jiné vědce je kompilátor Fortranu mnohem důvěryhodnějším optimalizátorem využití paměti než C++. Ukazatelé se ve Fortranu narozdíl od C++ nevyskytují.

2.3.3 Budoucnost Fortranu

Podle průzkumu uživatelů, kteří pracují se superpočítači, se bude Fortran používat ještě několik let. (software intelcom [13]) S jazyky C a C++ se Fortran používá k matematickým výpočtům, jiné jazyky nemají tak dobré rychlostní výsledky. Aby Fortran přežil, musí se odstranit jeho nedostatky převážně v oblasti problematiky velkých projektů. S tímto souvisí omezená kontrola typu, nedostatečná rozšiřitelnost, spoléhání se na globální data atd., tohle všechno činí Fortran

obtížný na údržbu a ladění, než je to v C++. Naproti tomu C++ začíná být velmi dobrým vyzyvatelem Fortranu v efektivnosti. Navíc C++ nabízí spoustu výhod převážně ve velkých projektech a to jsou **zapouzdření**, **přetížení operátorů**, **dědičnost** a **dynamická vazba**. Ukázka kódu ve Fortranu a C++. (Výpis 1 Fortran a Výpis 2 C++)

```
c Ukazka kodu ve Fortranu
c Dynamicke pridelovani a uvolnovani poli ve Fortranu. Alokovani 2D pole.

real, dimension(:,:), allocatable :: name_of_array
allocate(name_of_array(xdim, ydim))
```

Výpis 1: Dynamické přidělování a uvolňování polí ve Fortranu

```
{ V jazyce C++ , podobna zalozitost }
int ** array;
array = malloc(nrows * sizeof(double * ));

for(i = 0; i &lt; nrows; i++)
array[i] = malloc(ncolumns * sizeof(double));
```

Výpis 2: Dynamické přidělování vícerozměrných polí v C++

Tabulka 1: Tabulka srovnání rychlosti Fortran vs. C++

Fortran Intel versus C++ g++ fastest programs by faster benchmark performance			
pidigits			
source	secs	mem	cpu load
Fortran Intel	1.74	3.896	0% 2% 3% 100%
C++ g++	1.88	4,376	1% 3% 100% 1%
spectral-norm			
source	secs	mem	cpu load
Fortran Intel	1.98	1,684	99% 100% 99% 100%
C++ g++	1.98	1,164	99% 99% 99% 100%
n-body			
source	secs	mem	cpu load
Fortran Intel	8.28	8	100% 100% 100% 100%
C++ g++	8.10	1,084	100% 100% 96% 100%
fannkuh-redux			
source	secs	mem	cpu load
Fortran Intel	12.59	10,912	100% 100% 100% 100%
C++ g++	10.12	1,852	100% 100% 96% 100%
binary-trees			
source	secs	mem	cpu load
Fortran Intel	4.78	985,316	1% 100% 0% 0%
C++ g++	2.93	980,716	51% 23% 33% 51%
reverse-complement			
source	secs	mem	cpu load
Fortran Intel	4.78	985,316	1% 100% 0% 0%
C++ g++	2.93	980,716	51% 23% 33% 51%
fasta			
source	secs	mem	cpu load
Fortran Intel	3.34	8	0% 2% 100% 0%
C++ g++	1.33	1,740	81% 81% 81% 82%
mandelbrot			
source	secs	mem	cpu load
Fortran Intel	61,460	3.896	82% 82% 82% 100%
C++ g++	1.51	25,640	100% 100% 99% 99%

3 Program a hlavní činnosti mojí bakalářské praxe

3.1 Program RollFlex

Program RollFlex slouží k výpočtu tvaru válcovací mezery soustavy válců a příčného profilu válcovaných plochých vývalků. Program RollFlex umožnuje:

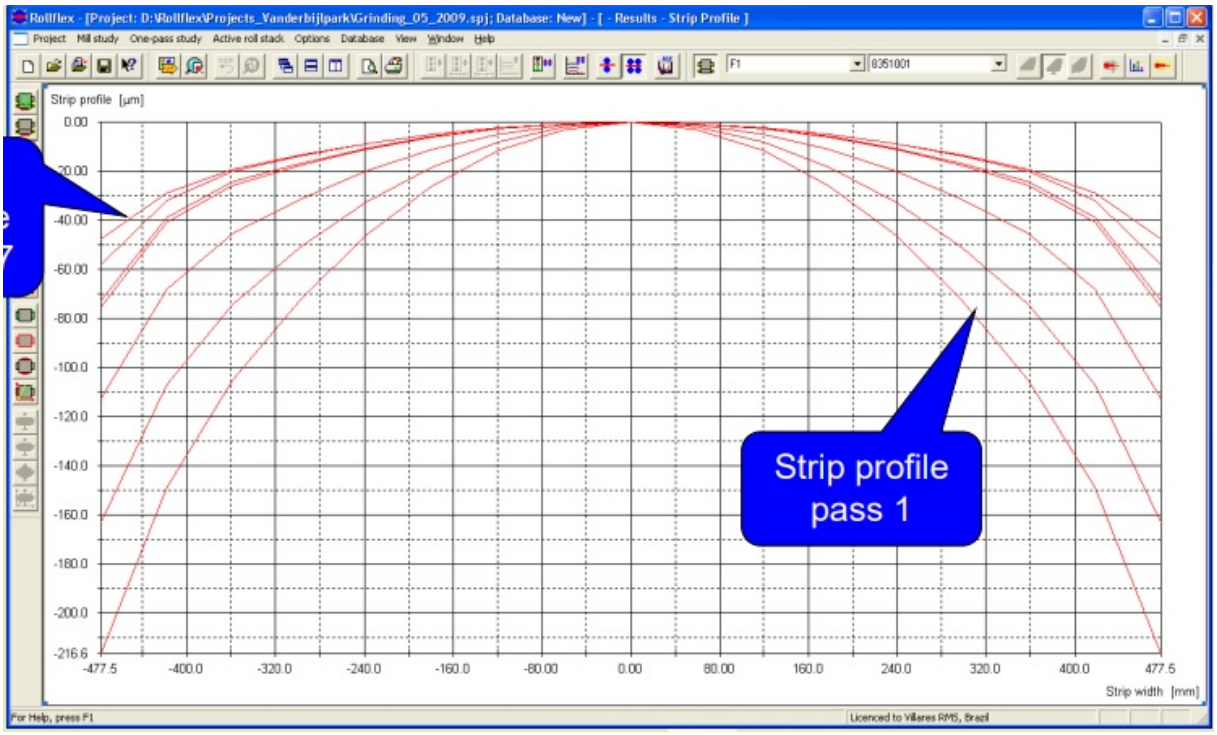
- výpočty přímého profilu plochých vývalků
- studium vlivu různých akčních členů pro různé typy stolic (duo, trio, kvarto, sexto):
  - protiohybové síly
  - axiální posun
  - výbrusy válců
- optimalizaci protiohybových sil, axiálního posunu a výbrusu válců
- zobrazení různých typů grafů se závislostmi příčného profilu, koruny a její citlivosti na různých technologických parametrech
- navržení speciálních výbrusů válců pomocí vestavěného generátoru výbrusů.

Shrneme-li si to, program slouží k válcování kovů a plechů pomocí válců a je rozšířen o funkce, které s tím souvisí. Jedná se například o predikci maximální délky trasy válcování a správného broušení, aby se zabránilo odlupování válců. Samotné válcování je dosti problematické a setkáváme se zde s mnoha problémy. Program se snaží těchhle problémů vyřešit co nejvíce. Jeden z nich můžeme vidět na obrázku (Obrázek 1).

Obrázek 1: Povrchové trhání nebo praskání válců

Jedny z těchto problémů jsou problémy se zatížením a napětím na povrchu válce nebo Hertzovo kontaktní napětí, což je tlak, který vzniká v místě vzájemného silového působení dvou těles s definovaným zakřivením povrchu. Dále ohybové napětí, zbytkové napětí a mnoho dalších.

Program k těmto problémům poskytuje grafy a matematické výpočty. Na obrázku (Obrázek 2) můžeme vidět uživatelské rozhraní programu RollFlex.



Obrázek 2: RollFlex a zobrazení pásových profilů

Pro bližší seznámení a jasnější pochopení problematiky by bylo vhodné popsat válcovací stroje detailněji. Bohužel moje znalosti na toto nestačí. Byl jsem na praxi seznámen pouze s programy, na kterých jsem pracoval. Většina matematických rovnic a pravá podstata fungování přesahuje běžné schopnosti, a proto se nedá do problematiky plně vžít.

3.2 Překlad K2, K3, K4 programů

Celkové využití programů je na výpočet příčinku a průhybu pracovního válce. Tyto programy mi byly předány na překlad z Fortranu do C++, modifikaci a optimalizaci.

K2 program slouží ke kontrole a načtení základních dat, ze kterých se vypočítává koeficient rozloženého zatížení a dále tyto data připravuje a třídí na přenos do programu K3.

K3 program se používa pro sestavení matice soustavy, pravé strany a vyřešení soustavy nelineárních rovnic pro styk dvou válců. Dále pro výpočet bombírování a pro další věci, které s tím souvisí.

K4 program slouží na výpočet příčinku a průhybu pracovního a opěrného válce a k předání výsledků do programu RollFlex.

Tyhle programy znázorňovaly několik funkcí v programu RollFlex. Všechny programy byly napsány ve Fortranu. Obsahovaly asi 10-30 tisíc řádků každý.

Mým úkolem bylo celé programy přeložit z Fortranu do C++ a to tak, aby byly odstraněny všechny chyby, které se v programech vyskytly - programátorské i výpočetní. Dalším mým úkolem bylo sloučit tyto programy a naimplementovat je do RollFlexu. Nakonec optimalizovat tyto programy, aby si předávaly informace přes paměť a ne pomocí textových nebo binárních souborů.

3.2.1 Problémy s přepisem

První problém, na který jsem narazil, bylo záporně definované pole ve Fortranu, které se chovalo trošku jinak než normální pole. Důležité je si sdělit, že v C++ záporné pole není. Řešení bylo následující: porozumnění logice části kódu se záporným indexováním pole a znovunapsání kódu v C++ se stejnou funkčností. Kód se zkrátil asi o 50%.

3.2.2 Minoritní komplikace

Jedním z malých problémů bylo špatné indexování. Ve Fortranu indexování začínalo od 1, v C++ je to standartně od 0. Mohl jsem ponechat indexování Fortranu a prostě přepsat kód jedna k jedné do C++. Indexování od 1. mně nepřišlo příliš vhodné a jenom by zhoršilo čitelnost a zároveň obtížnost debuggování při běhu programu. Tyhle posuny cyklů a polí mně zabraly několik hodin. Výsledkem byl čistší kód.

3.2.3 Debug vs. Release

U projektu se průběžně zkoušela jeho funkčnost s různými daty, a proto se průběžně uváděl do provozu. Musel být funkční jako .exe soubor. Tady jsem se poprvé setkal s prací a debuggováním v *release* režimu.

- Debug : při kompilaci v debug režimu dostaneme ".pdb"soubory společně s .exe soubory nebo .dll ve standartním nastavení. Tyhle soubory .pdb se nazývají "symboly"a obsahují data, která nám umožnují trasování za běhu programu. Uchovávají také záznamy o proměnných a tak pomocí *watcherů* se můžeme podívat do konkrétní proměnné, co skrývá za hodnoty. Při vyhození chyby nám říká, ve které třídní metodě nebo funkci nastala chyba. Dokonce nám umožňuje zobrazit konkrétní řádek, kde se chyba odehrává. Dále umožňuje schopnost přeskočit daný kód atd.

- Release : při kompilaci v *release* režimu kompilátor optimalizuje zkompilovaný kód, aby při spuštění byl co nejvíce efektivní. V *release* režimu se kompiluje trošku odlišněji než v debug režimu. Pokud je vyhozena výjimka, nedojde k přesnému upozornění, ve které třídě je chyba. Dokonce ani breakpoint nejde nastavit tak, aby se kód zastavil na patřičném řádku. Je mnohem přísnější co se týče samotné kompilace. Neinicializované proměnné vám vyhazují chyby, v mém případě kompilátor rozeznal potíže s příkazem *goto*. Tento příkaz

přeskakoval deklaraci některých proměnných v kódu. Další věcí je, že v *debug* režimu vám projde dělení nulou, v *release* nikoliv. (Stephen Prata, Mistrovství v C++ [1])

Režimy jsou do určité míry nastavitelné programátorem. Standartně Visual Studio nastavuje režim *debug* pro ladění výstupů. Je možné přenastavení *release* režimu, tak aby zachytil breakpointy a další věci. (Joel Murach [3])

Během překladu jsem narazil na spoustu chyb, které se projevovaly pouze v *release* režimu. Většina z nich zapříčinila, že Visual Studio je přísné na špatné zachazení s příkazem *goto*. Narazil jsem na nahrávání dat mimo pole, kde v *debug* režimu moje funkce počítala správně, ale v *release* nikoliv. Taky jsem narazil na zvláštní případy, kdy jsem v cyklu dělil za určitých podmínek nulou - *debug* režimu to nevadilo, ale v *release* režimu to vyhazovalo chybu. Tyhle chyby se špatně hledají, protože vám kompilátor většinou vůbec nic neoznačí.

Jak už jsem zmínil, tyhle problémy se řeší špatně. Proto jsem vymyslel log systém, který nedělal nic jiného, než že vypisoval proměnné do souboru na místech, kde jsem si myslel, že by mohla být chyba. Pomocí tohoto systému jsem odstranil všechny potíže v releas režimu.

3.2.4 Proč nepoužívat Goto

Jeden z dalších problémů, se kterým jsem se setkal, bylo řešení příkazu *goto* a jeho nesprávné použití. Nejdříve si povíme něco o samotném příkazu.

3.2.4.1 Příkaz Goto

- Příkaz *goto* přenáší chod programu na místo určené takzvaným lablem. Důležité je, aby byl ve stejné funkci nebo metodě jako je label. Chod programu může být přemístěn za i před *goto* příkazem.
- Pokud přenos opustí nějaký blok kódu, kde byly vytvořeny proměnné, může se stát, že tyhle proměnné budou smazány a také se může stát, že můžeme přeskočit nějakou část kódu, kde naopak mají být vytvořeny proměnné, se kterými pracujeme.
- Dále se neumožňuje přenést do bloku *try* nebo *catch*.
- Kompilátor tyto všechny problémy nedokáže odhalit a nastávají errory během běhu programu.

  (From Beginner to Expert on Udemy [11])

Můj úkol byl prostý: odstranění veškerých *goto* příkazů v kódu. Musel jsem dobře porozumnět funkčnosti kódu, abych byl schopný ho přepsat do C++ bez použití příkazu *goto*. Setkal jsem se s jednoduchými případy, kdy byl ve Fortranu použit příkaz *goto*, aby nahradil obyčejný "Do while"cyklus, který se ve Fortranu jinak řešit nedal. Ve většině případů jsem nahrazoval *goto* složitější "if"podmínkou nebo cyklem. V některých případech musel být kód ponechán v

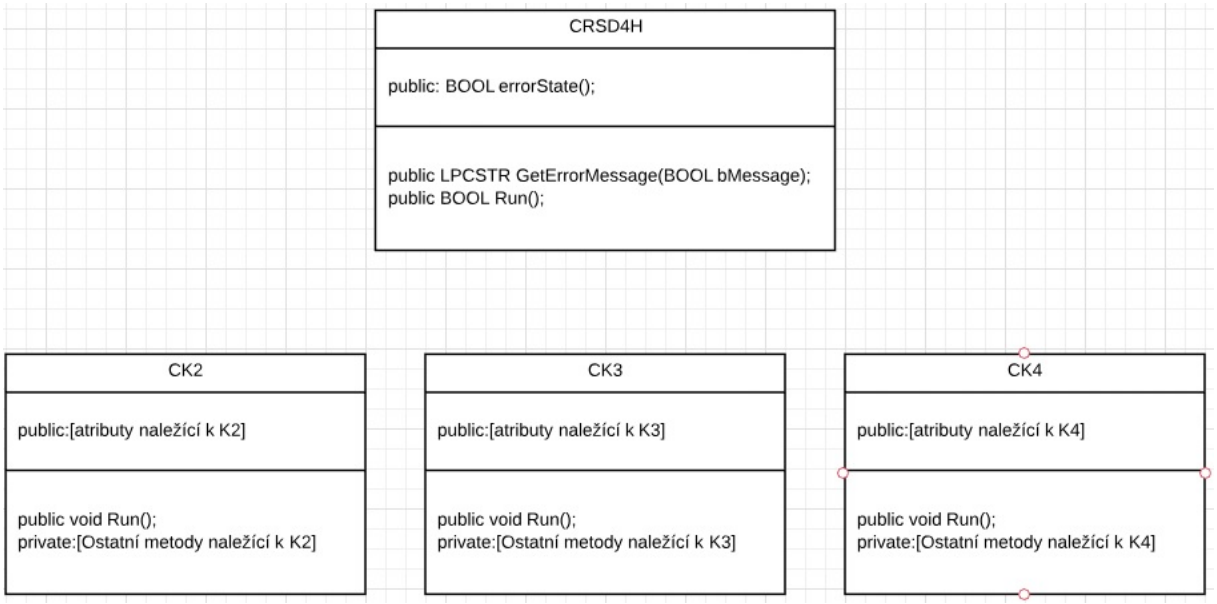
původním stavu, jak můžeme vidět v následujícím ukázce kódu (Výpis 3). Je zde znázorňen jeden z případů, které jsem nechal v původním stavu.

```
{ Typicky priklad jak Goto znici  citelnost  }

 if  (abs(FOUR − T[1][LM1]) <= H2TOL) goto g60;
 if  (T[1][LM1] == ZERO) goto g55;
 if  (abs(TWO − abs(T[1][LM1])) < JUMPTL) goto g130;
 if  (L == 3) goto g15;
H2CONV = false;
 if  (abs((T[1][LM1] − T[1][L − 2]) / T[1][LM1]) <= AITTOL) goto g75;
 if  (REGLAR) goto g55;
 if  (L == 4) goto g15;
HRERR = ERGL + ERRER;
g55:
 if  ((ERRER > ERGOAL) && (HRERR != ERGL)) goto g175;
goto g145;
//                                   CAUTIOUS ROMBERG EXTRAPOLATION
g60:
 if  (H2CONV) goto g65;
AITKEN = false;
H2CONV = true;
g65:
FEXTRP = FOUR;
g70:
IT = IT + 1;
VINT = STEP ∗ T[L][IT];
ERRER = abs(STEP / (FEXTRP − ONE)∗T[IT − 1][L]);
 if  (ERRER <= ERGOAL) goto g160;
HRERR = ERGL + ERRER;
 if  (HRERR == ERGL) goto g160;
 if  (IT == LM1) goto g125;
 if  (T[IT][LM1] == ZERO) goto g70;
 if  (T[IT][LM1] <= FEXTRP) goto g125;
 if  (abs(T[IT][LM1] / FOUR − FEXTRP) / FEXTRP < AITTOL) FEXTRP = FEXTRP ∗
     FOUR;
goto g70;
```

Výpis 3: Zhoršení čitelnosti a kvality kódu u příkazu Goto

3.2.5 Sloučení programů

Po dokončení překladu následovalo samotné sloučení programů. Doposud si programy předávaly informace za pomocí souborů, tuhle komunikaci jsem chtěl odstranit a zároveň je předělat do tříd. Tento krok slouží k 100% využití potenciálu C++. Prvním krokem bylo vymyslet jednoduchý systém neboli strukturu programu. Vytvořil jsem z každého programu (K2, K3, K4) třídu a pak ještě jednu třídu (CRSD4H), která obsahovala jenom jednu metodu Run. V téhle metodě jsem vytvářel instance tříd K2, K3 a K4. Třídy K2, K3 a K4 obsahovaly také jednu metodu, která se starala o vlastní chod programu. Zjednodušeně řečeno, každý z přeložených programů měl jednu hlavní metodu, která se stará o běh jednotlivého programu. S tím, že metoda Run() třídy CRSD4H se starala o komunikaci se třídami K2, K3 a K4. Pro představu nadcházející obrázek a ukázka kódu. (Obrázek 3 a Výpis 4)



Obrázek 3: Zjednodušený diagram tříd

3.2.6 Problémy se sloučením programů

Po odstranění čtení ze souboru začaly komunikovat programy pomocí paměti, ale nastal zde problém, se kterým jsem se ještě nikdy nesetkal - "Run Time Check Failure #2 - Stack Around The variable 'x' was corrupted"viz. obrázek (Obrázek 4). Run-Time Check Failure byl způsoben tím, že jsem pracoval s příliš velkým množství operační paměti. Jednoduše řečeno, program byl příliš velký a náročný na výpočet a v půlce běhu spadnul.

Řešení bylo následovné. Nejdříve jsem hodně statických proměnných změnil na pointery a následně je hned mazal, když už neměly v programu využití. Tímhle řešením byl však jen problém oddálen. Nakonec mě napadlo že vytvořím pointer na celou instanci třídy. Jakmile jsem s instancí přestal pracovat, přiřadil jsem jí NULL a následně smazal. Pro ukázku výpis kódu (Výpis 5).

Problém se z části vyřešil, ale místo něho se objevil jiný podobný error. Domnívám se, že to bylo z jiného důvodu, nejsem jsem si však jistý, v čem byl problém. Nahrával jsem do ukazatele hodnoty a v půlce tohoto procesu nahrávání se objevovala chyba. Opravil jsem to následujícím způsobem: z proměnných jsem udělal třídní proměnné a chyba se opravila. Zkoumal jsem v čem byl problém, ale na nic jsem nepřišel. Výsledkem byl sloučený program K2, K3 a K4 připravený na implementaci do RollFlexu.

```
{ Main v programu vytvarim instanci tridy CRSD4H ktera ridi beh K2..K4 }

int _tmain(int argc, _TCHAR* argv[])
{
    BOOL error;
    CRSD4H * rsd4h = new CRSD4H();
    error = rsd4h->Run();

    delete rsd4h;
    rsd4h = NULL;
}
```

Výpis 4: Main kod

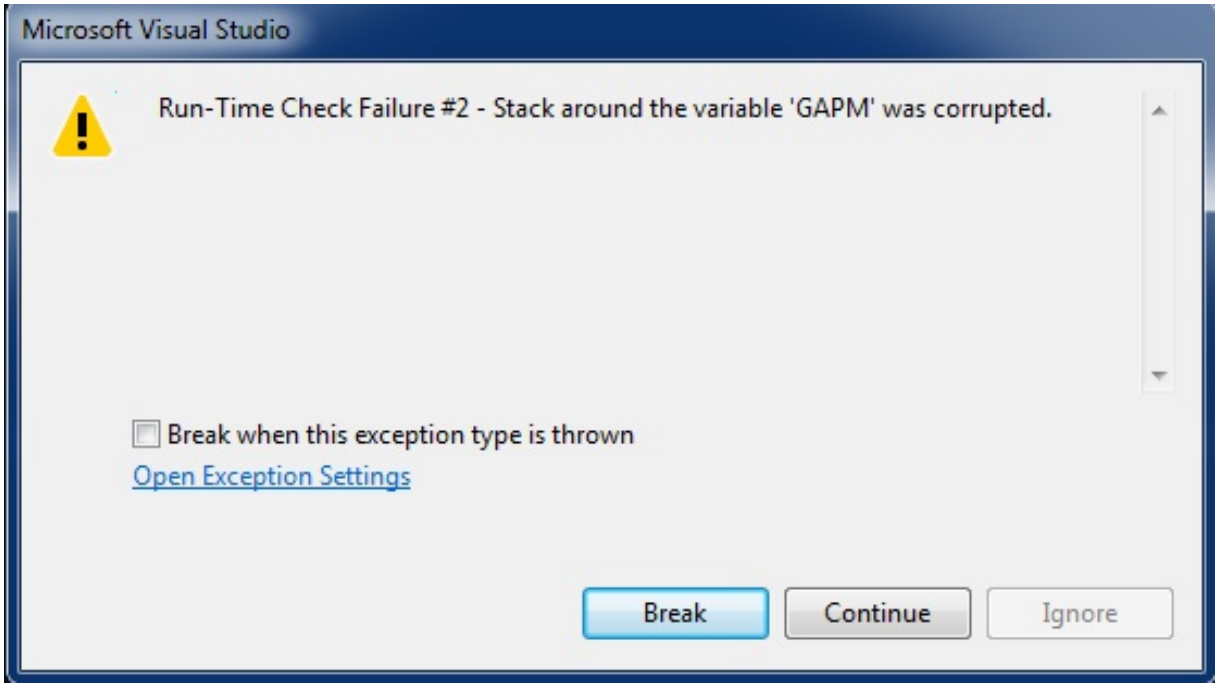
```
{ Metoda Run spravuje K2, K3, K4 tridy. }

BOOL CRSD4H::Run()
{
   CK2* k2 = new CK2();      // důležité je že instance tříd jsou ukazatele.
   try{
      k2->Run();             // try bloku pokud nastane chyba ,tak zachytím.
   }catch(BOOL errorState){
      return = errorState        // vrátím chybu jako BOOL (představuje int).
   }

   CK3* K3 = new CK3(k2->parametr1, k2->parametr2 . . . . . .);
   delete k2;                 // až není za potřebý hnedka se maže.
   k2 = NULL;                 // pro jistotu pointer nastavím na NULL tím se nic nezkazí.
   try{
      k3->Run();
   }catch(BOOL errorState){
      return = errorState
   }
// . . . atd
}
```

Výpis 5: Metoda Run()

3.2.7 Error systém

Kromě standartního výstupu, který poskytoval můj program programu RollFlex, poskytoval stav, ve kterém program skončil. Pokud skončil úspěšně a pokud došlo k chybě, například k přehřátí materiálu, a program se musel pozastavit, tak vždy předával informaci o stavu programu RollFlexu. Bylo zde asi 30 možných variant, jak program mohl skončit. Ne vždy se jednalo o vypnutí celého programu. V některých případech došlo k pozastavení nebo předání neúplného výsledku a následně k vrácení na místo, kde se program zastavil. Někdy šlo jenom o varování, které se pak zobrazovalo uživateli jako výstup.

Nejdříve jsem vytvořil Header soubor, který obsahoval všechny možné eventuality chyb. Pomocí #define "ErrorName""number"jsem definoval všechny chyby. Dále jsem vytvořil metodu ve třídě CRSD4H GetErrorMessage (BOOL bMessage), která vracela LPCSTR. LPCSTR , string a CString jsou dosti podobné stringu v C#, ale jsou mezi nimi malé rozdíly. Nám však postačí pouze tato skutečnost.



Obrázek 4: Run Time Exception

Metoda nebyla nic jiného, než jeden velký *switch* příkaz, který podle vyhozené výjimky ukončoval můj program a předával výsledek RollFlexu. Pomocí *Try* a *Catch* bloků, které obalovaly řídící metody patřičných tříd, jsem chybu vracel. Pro přehled (Výpis 5). Nešlo o nic jiného, než o systém, který se stará o nestandartní ukončení programu.

3.2.8 Výsledek práce

Výsledná práce nahradila všechny .exe soubory, který dříve pracovaly s RollFlexem. Celý program se stal jeho součástí a byl implementován do RollFlexu. Už nepracoval se soubory, nýbrž si předával informace pomocí paměti. Celý kód byl napsán v C++. Zlepšila se čitelnost, rychlost a opravily se nedostatky programu. Celkově se sním pracovalo lépe. Hlavním přínosem bylo, že se kód stal srozumitelný pro ostatní ve Firmě a mohli ho upravovat podle libosti.

3.3 Prezentační program

3.3.1 Knihovna MFC

MFC je objektově orientovaná knihovna v C++. Je určená pro vyvíjení desktopových aplikací pro Windows. MFC byla představena veřejnosti firmou Microsoft v roce 1992. V podstatě MFC je SDK rozhraní. Je to jedna velká knihovna, která se skládá ze sady tříd, které pomáhají programátorovi vytvářet rozhraní API systému Windows. Kompilátor Visual Studio dokáže automaticky vygenerovat samotnou kostru MFC kódu na použití v projektu a tím ulehčit pro-

gramátorovi práci. MFC byla z počátku vytvářena ve věku, kdy se ještě hodně programovalo v C, a proto je hodně kódu naprogramováno v C. (Alan R. Feuer, MFC programing [4])

3.3.2 O aplikaci

Mým cílem bylo vzít kousek funkčnosti z mého finální projektu a udělat pro ní desktopovou prezentační aplikaci. Taková aplikace měla sloužit jako předváděčka pro poptávající po válcových systémech. Co se týče grafického návrhu jsem měl svobodnou tvůrčí práci. (Obrazek 6) Jediný požadavek byl na její funkčnost. Měl jsem zobrazit výpočet liniové síly a profilu pásu provalku v grafu. Na vykreslování bodů v grafu měla firma ITA napsanou svoji knihovnu (GphCtls). Tuhle knihovnu jsem si upravoval a pracoval sní.

3.3.3 Vývoj

Nejprve jsem si musel uvědomit, které všechny atributy budu potřebovat na samotný výpočet, který by se měl odehrávat před zobrazením grafů. Vzal jsem dvě funkce, které jsem překládal z mého finálního programu. Tyto funkce jsem vložil do nově vytvořené MFC aplikace. Vytvořil jsem pro ně samostatný projekt, ve kterém jsem vytvořil třídu pro moje funkce. Následně jsem je nalinkoval s MFC projektem a ověřil jejich funkčnost s rozumnými hodnotami.

Poté přišlo na řadu vytvořit grafické rozhraní, abych nemusel zadávat atributy v kódu, ale aby si je mohl zvolit uživatel sám. Nejdříve jsem si vzal tužku a papír a načrtnul několik verzí jak by mohlo GUI vypadat. Nakonec jsem si rozmyslel, že pro tak jednoduchou aplikaci bude nejlepší nějaký standartní jednoduchý vzhled. Dalším krokem bylo moje načrtlé GUI realizovat. MFC funguje na principu "Drag and drop", což je obyčejné tahání tlačítek a oken z pomocné lišty. Podobně jako u Windows form application ve visual studiu. U každé proměnné jsem musel udělat speciální výjimky. Když uživatel zadal nesmyslný parametr tak výjimku zobrazit na obrazovku. Také jsem zabudoval logový systém neboli informační logové okno, které informovalo uživatele o tom, co právě udělal za akci.

Když jsem vyvíjel aplikaci tak jsem udělal spoustu chyb a uvědomil jsem si některé záležitosti co se týče správně udělaného GUI. Například jsem chyby zobrazoval jenom do logového systému. Místo, abych zobrazil chybu na speciálním windows okně, které vyskočí do popředí, když uživatel způsobí chybu. Dále moje aplikace obsahovala dvě tlačítka jedno pro vykonání výpočtu a druhé pro uložení takových výpočtů do souboru. Ukládaní do souborů jsem už dělal v předcházející části mojí práce. Vzal jsem dosavadní kód a upravil ho tak, aby vyhovoval dané situaci. Nejtěžší částí bylo zobrazování grafů a práce s knihovnou.

3.3.4 Práce s knihovnou

Knihovna, se kterou jsem měl pracovat, se jmenovala GhpCtls 32. Na práci s touhle knihovnou jsem dostal programátorskou příručku v PDF formátu.(Obrazek 5) Programátorská příručka ke knihovně nebyla aktualizována a pracovalo se sní velmi nepohodlně. Obzvlášť, když nevíte tenhle

fakt a počítáte, že je správná. Jinak příručka obsahovala jak integrovat GphCtls 32 knihovnu do aplikace, inicializace dat pro 2D-grafiky, pro 3D věci a další vlastnosti grafů. Už při integraci knihovny do projektu jsem měl dost velké potíže. Nastavoval jsem linker tak aby knihovna .lib byla slinkovaná s aplikací a to podle následujících kritérií:

- GphCtls32d.lib debug verze, singlethreaded,
- GphCtls32MTd.lib debug verze, multithreaded, knihovna MFC je linkována staticky,
- GphCtls32RTd.lib debug verze, multithreaded, knihovna MFC je linkována dynamicky,
- GphCtls32.lib release verze, singlethreaded,
- GphCtls32MT.lib release verze, multithreaded, knihovna MFC je linkována staticky,
- GphCtls32RT.lib release verze, multithreaded, knihovna MFC je linkována dynamicky.

Musel jsem přidat do hlavičkového souboru StdAfx.h hlavičkový soubor knihovny GphCtls32.h a přidat do prostředků (resources) projektu soubor s prostředky knihovny GphCtls32CSY.rc (česká verze) a hlavičkový soubor s konstantami pro tyto prostředky GphRes.h. Nakonec zavolat funkci při spuštění GcInitialize a při ukončení funkci GCUninitialize. Pokaždé, když jsem našel rozdíly mezi programátorskou příručkou a knihovnou, tak jsem ji aktualizoval pro budoucí uživatele. Další práce s knihovnou už byla jednoduchá.

2. Inicializace dat 2D-grafiky

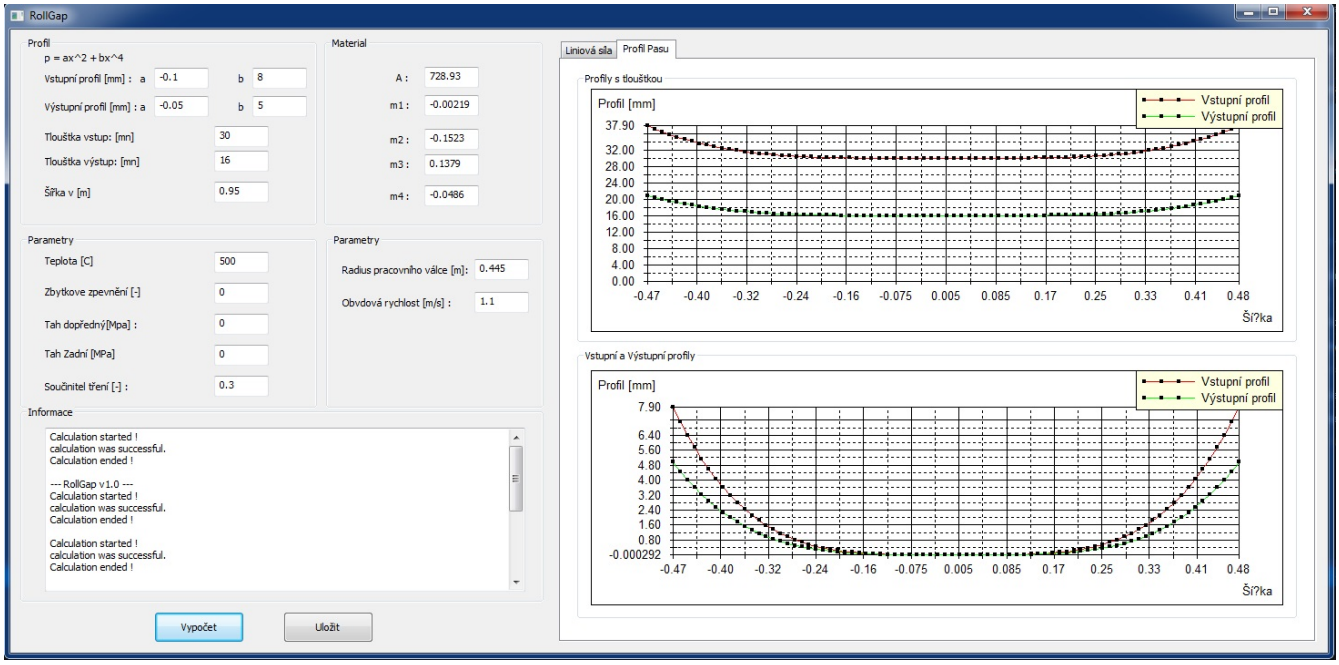
Posloupnost kroků potřebných pro inicializaci 2D-grafiky:

a) získání ukazatele na instanci třídy **C2DGraphSet** (základní třída pro práci s 2D-grafikou) – metoda **GetGraphSet** tříd C2DView a C2DGraphWnd,
b) vytvoření minimálně 1 instance třídy **C2DGraphSubSet** (podmnožina grafů; Všechny grafy sdílejí společnou osu *x*, v ose *y* se mohou lišit. Instance třídy C2DGraphSubSet rozdělují grafy podle osy *y*) – **new C2DGraphSubSet**,
c) inicializace dat grafu – naplnění struktury **_2D_FUNCTION_CREATE_STRUCT** (inicializace funkcí) nebo **_2D_POINTS_CREATE_STRUCT** (inicializace ostatních typů grafů),
d) přidání grafu do některé instance třídy C2DGraphSubSet podle příslušnosti k ose *y* – metody **AddFunction** a **AddPoints** třídy C2DGraphSubSet

Obrázek 5: Útržek z programátorské příručky

3.3.5 Výpočet koordinantů

Dále bylo potřeba vypočítat koordináty z atributů, které zadával uživatel. Pomocí příručky ke grafům zavolat patřičné metody pro inicializaci grafů a následně pro vykreslení koordinátů. Pak jsem jenom upravoval samostatný vzhled grafů.



Obrázek 6: Prezentační program v MFC C++

3.4 Modifikace stávajícího programu

Během praxe jsem přidával a modifikoval program RollFlex o různé vlastnosti a funkce, které vypomáhaly s výpočtem a nebo zvětšovaly čitelnost programu. Většina z nich byly matematické rovnice a podpůrné programy pro výpis dat nebo různé třídění či úpravu dat.

3.4.1 Řešení soustavy lineárních rovnic pro N rovnic

Soustavu jsem potřeboval řešit pro program, který počítal zatížení jednotkového příčinku. Tento program přebíral vstupy jako vektor pravé strany, počet rovnic a matici soustavy. Mým úkolem bylo tuto soustavu vyřešit. Řešení jsem prováděl pomocí Gaussovy eliminační metody. Program jsem vytvořil za pomocí návodů z internetových zdrojů (Stack overflow [7], Programujte.com [6]) a modifikoval ho podle mých potřeb. Pro představu zkrácený výpis kódu. (Výpis 7)

3.4.2 Funkce na čtení špatně formátovaného souboru

Tato funkce vznikla kvůli špatnému čtení ze souborů. Program K2 jako jediný musel číst data z textových souborů. Tato vstupní data se měnila podle prováděného výpočtu. Jednoduše pro jeden výpočet existovalo více vstupních souborů, které byly vybírány podle úlohy. Tyto soubory se lišily formátováním. Místo několika funkcí na každé čtení jsem udělal jednu funkci, která dokázala přečíst všechny soubory, bez ohledu na jejich odlišné formátování. V souborech se velmi nepravidelně objevovaly komentáře, jména proměnných, vědecky zapsaná čísla a různé znaky. Někdy to byl soubor plný *doublů*. Vytvořená funkce tento problém řešila obecně pro všechny

případy. Výsledkem byl zkrácený kód, zvětšení čitelnosti a hlavně vytvoření znovupoužitelného kódu pro nastávající programová řešení ve firmě.

3.4.3 Třída CustomLog

Třída byla vytvořena z potřeby zlepšení debugování v *realese* režimu. Obsahuje několik metod, pomocí kterých jsem vypisoval informace z proměnných do souborů. Metody byly přetížené, tak abych mohl vypisovat všechny možné typy proměnných. Jednalo se o pomocnou třídu, která mě usnadňovala formátování výpisů a tím zlepšovala rychlost hledání chyb. Pomocí této třídy jsem vyřešil mnoho problému, které se vyskytovaly v *realese* režimu. Výpis kódu (Výpis 6) ukazuje některé metody ve třídě.

```
{ Ukazka metod tridy CustomLog }

void CCustomLog::Add(double value)
{
    CString cPom;
    cPom.Format(_T("%f"), value);
    Add(cPom);
}

void CCustomLog::AddSeparator()
{
    if (this == NULL)
    return;

    if (!m_bEnableLog)
    return;

    fstream LogFile(m_sPath, fstream::app);
    CString line;

    for (int i = 0; i < 150; i++){
    line += _T("-");
    }

   .............
```

Výpis 6: CustomLog Class útržek kódu

```
{ Vypocet rovnic zapomoci Gaussovy eliminacni metody. }

#include "K3.h"
/*
  Name: AdGaussLS
  Function: solves the overdetermined system B*x=c of linear algebraic equations

  Autor: ITA s.r.o,
  Modifications:
*/
int CK3::AdGaussLS(int M, /* Number of equations */ int N, /* Number of unknowns */
double **B, /* System matrix of (MxN) */ double *c, /* Right hand side vector (Mx1) */
double *x) /* Solution (Nx1) (OUTPUT) */
{
   double TOL_SINGULAR = 1.e-10;
   int k = 0, i = 0;   double   *a1, *a2, t;
   double *A, *d; // LSq system A*x=d (A=[row1,row2,row3,...,rowN] of type NxN)

   if (M < N) return 1; // ERROR: System must be overdetermined or square!

   // Initialization
   A = new double[N*N]; d = new double[N];
   for (i = 0; i < N; i++) {
      for (int j = 0; j < N; j++) A[i*N + j] = 0;
   d[i] = 0;
   }

   // Assembling A, d
   for (i = 0; i < N; i++)
      for (int k = 0; k < M; k++) {
         for (int j = 0; j < N; j++) A[i*N + j] += B[k][i] * B[k][j];
      d[i] += B[k][i] * c[k];
   }

/* Atd . . . . . */
```

Výpis 7: Gaussova eliminační metoda

4 Závěr

Nejprve bych chtěl říct, že jsem program nedokončil. Skončil jsem na 90% vývoje. I když jsem projekt nestihl dokončit, i přesto jsem ho vypracoval do funkčního stavu. Celkový běh programu se zrychlil. Největší vliv na rychlost měla práce se soubory. Jakmile jsem začal předávat informace pomoci paměti, tak se program zrychlil o několik sekund.

Porovnání staré verze a mého finalního projektu v sekundách		
Prog. Language	Debug	Release
Fortran	13.74	3.421
C++	5.405	0.854

4.1 Schopnosti a znalosti použité v průběhu praxe

V průběhu praxe jsem uplatnil především znalosti v C++. Tyto znalosti jsem nabyl na vysoké škole a samostudiem během studia. V předmětech jako Programování I, Programování II, dále Algoritmy I, Algoritmy II a dalších předmětech, které mě naučily základy o dědičnosti a jak používat návrhové vzory.

4.2 Schopnosti a znalosti chybějící v průběhu praxe

Nebylo téma, kde mě znalosti nechyběly. Nejvíce mě chyběly znalosti o knihovně MFC a o jazyku Fortran, které pro mě byly úplně nové. Dále znalosti týkající se všeobecného nastavení Visual Studia, práce s verzovacími programy jako je GitHub a také pokročilejší znalosti C++.

4.3 Celkové shrnutí

Během vývoje jsem měl hodně problému a tak jsem se naučil spoustu užitečných věcí. Něco málo ve Fortranu a především programování v C++. Taky práci ve Visual Studiu a snad nejvíce přínosnou částí byla práce v týmu, sdílení kódů s ostatními, komunikace ve firmě, vztahy mezi zaměstnanci a vztah k mému nadřízenému.

Jsou věci, které nás ve škole neučí a to je především řešení chyb. Učíme se hodně teoretických věcí, a když přijdete do praxe, realita je úplně někde jinde. Většinu věcí se musíte naučit až v ten moment, kdy jsou doopravdy zapotřebí a pak se teorie přesune i do praxe. Takové věci jako linkování knihovny a nebo rozdíly mezi *release* a *debug* režimy a jejich výhody a postup při debugování *release*. O pořádné optimalizaci jsme mluvili jenom teoreticky, ale nikdy jsem nic takového ve škole nedělal. Tohle všechno a další věci jsem se naučil až v praxi. Popravdě po zkušenostech z praxe si myslím, že by bylo výhodné zavést více povinné praxe do studia.

Moji praxi hodnotím pozitivně. Jsem rád, že jsem chodil na praxi a nedělal jenom bakalářskou práci sám. Dokonce si myslím, že zrovna pro člověka mého typu je praxe přínosnější než studium.

Mým názorem je, že všichni studenti by měli co nejdříve alespoň na jeden nebo dva dny v týdnu pracovat v oboru, který je zajímá. Z mého pohledu jsou zkušenosti k nenahrazení.

Literatura

[1] Stephen Prata, Mistrovství v C++ 4. aktualizované vydání.

[2] Robert Lafore, Object-oriented programming in Microsoft C++. Rok vydání: 1991

[3] Joel Murach, Mary Delamater, Murach's C++ Programming. Rok vydání: 2018

[4] Alan R. Feuer, MFC programing, Covers C++ programing for Windows, Focuses on the Hardest topic of MFC.

[5] Learning Fortran , TutorialPoint simplyeasylearning. Dostupný na https://www.tutorialspoint.com/fortran/

[6] Stack overflow. Dostupný na https://stackoverflow.com/

[7] Programujte.com Dostupný na http://programujte.com/

[8] Computer Langue Benchmarks. Dostupný na https://benchmarksgame-team.pages.debian.net/benchmarksgame/faster/ifc-gpp.html

[9] More is Different. Dostupný na http://moreisdifferent.com/2015/07/16/why-physicsts-still-use-fortran/

[10] Kurzy na Udemy: Learn Advanced C++ Programming, Dostupny na: https://www.udemy.com/learn-advanced-c-programming/.

[11] C++: From Beginner to Expert, Dostupny na https://www.udemy.com/video-course-c-from-beginner-to-expert/

[12] Beginning C++ Programming - From Beginner to Beyond, Dostupný na: https://www.udemy.com/beginning-c-plus-plus-programming/

[13] Software Intelcom na https://software.intel.com/en-us/blogs/2015/03/27/doctor-fortran-in-the-future-of-fortran

[14] ITA spol. s r.o. [online]. Dostupné na http://www.itatech.cz/cs